Transfer Learning

Alan Casey and Alex Racapé

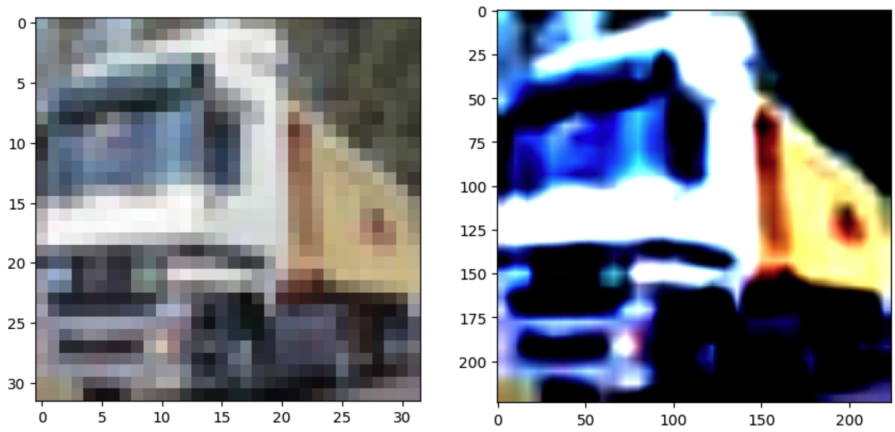
September 2023

1 Introduction

In this study, we delved into the realm of transfer learning—a powerful technique in machine learning that allows us to take advantage of pre-trained neural network models to solve new tasks efficiently. The primary objective of this experiment was to explore the efficacy of transfer learning in the context of classifying images from the CIFAR10 dataset. We leveraged two different neural network architectures, namely Resnet and VGG, and conducted studies to evaluate their efficiency and overall performance. We developed some hypotheses and even tested the models on images outside of the CIFAR10 dataset. We ultimately found little variation in the test accuracy for the different models, but there were significant costs for the more complex models in terms of training time.

2 Methodology

For this experiment, we utilized the CIFAR10 dataset, which is comprised of 60,000 32x32 pixel RGB images across ten distinct classes. 50,000 images are allocated for training and 10,000 for testing purposes. However, we cut down the number of training images to 24,000 in order to speed up training times. We pre-processed the images to fit the ImageNet standards, by resizing the images into a 224x224 pixel size and normalizing the pixel values. This code can be found on line 44 in the appendix, and Figure 1 depicts an example image from the truck class with and without our pre-processing transformations.



(a) Initial Image (b) With Preprocessing

Figure 1: CIFAR10 Example Image

To classify CIFAR10 images we made use of transfer learning. Transfer learning is a machine learning technique that involves adapting powerful pre-trained models to a new application. Instead of training the entire neural network from scratch, we can fine-tune the pre-trained model for our specific classification task. This approach often leads to faster convergence and improved accuracy. In this context, we classified CIFAR10 images with large models that were trained on the ImageNet database.

In our model selection, we considered four neural network architectures: Resnet50, Resnet152, VGG16, and VGG19. We used the pre-trained weights for each of these architectures, yet we fine-tuned them in order to perform image classification on the CIFAR10 dataset. To do this, we incorporated a Multi-Layer Perceptron (MLP) classifier after the models' respective feature learning layers. This code can be found on lines 159-164 for VGG models and 191-196 for our ResNet models. This MLP consisted of one hidden layer, with 128 neurons, a dropout layer, and an activation function before it reached the final output layer which has a total of 10 neurons for ten classes (airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks).

Our optimization process relied on ADAM, a stochastic optimizer that uses the benefits of momentum and adaptive learning rates (lines 169 and 201), with a learning rate of 1e-3. We also used the Cross Entropy (lines 168 and 200) loss function to measure the difference between our predicted and actual class labels during training. Additionally, all models were trained for 5 epochs on a T-4 GPU.

Our performance assessment centered on three critical metrics: total training, final classification accuracy, and inference time which measures the time it took the model to classify our 10,000 test images. We hypothesized that VGG19 and ResNet152 would take longer to train and test when compared with the simpler models. However we predicted that these models would achieve higher

(a) Frog 1 (b) Ship 1 (c) Plane 1

(d) Frog 2 (e) Ship 2 (f) Plane 2

Figure 2: Test Images from Google

accuracies. We also thought that the models using ResNet would outperform the VGG models because of it better performance on ImageNet. ResNets also typically require less work at the classification level.

To test the models on outside data, we also selected two random images from the internet that represent one of these three classes: planes, frogs, and ships. This experiment can be found on line 212 in the code appendix. These real-world images assessed the models' classification capabilities beyond the training dataset, and we measured each of the models' abilities for classifying new images. We aimed to have an easy image along with a slightly harder version for each class. A sample of some our images can be found in Figure 2.

3 Results and Discussion

Table 1: Model Comparison

	VGG16	VGG19	ResNet50	ResNet152
Training Time	12m 47s	20m 1s	12m 23s	22m 4s
Inference Time	1m 4s	1m 39s	59s	1m 45s
Accuracy (%)	81.87	81.64	80.66	81.93

Table 1 shows the differences in training time, inference time, and accuracy

across the four models. While there are slight differences in accuracy, all of the models were roughly even. ResNet152 had the best test accuracy, but it was at the cost of training and inference time. There were clear differences across the models in these times. For VGG and ResNet models, increasing the number of layers resulted in higher training and inference times. One notable difference was ResNet50 was the fastest to train and test with nearly a 50% improvement when compared to ResNet152, and its accuracy was only 1% worse after 5 epochs. The differences were similar for the VGG models as well. We are unsure why VGG19 had a lower accuracy than VGG16, but this difference may be insignificant and due to chance. Perhaps there would be more salient differences in accuracy if we were to train for a greater number of epochs.

Table 2: Image Tests

Image	VGG16	VGG19	ResNet50	ResNet152
Frog 1	✓	✓	Cat	✓
Frog 2	✓	✓	Bird	Bird
Ship 1	✓	✓	Plane	Plane
Ship 2	✓	✓	Plane	Plane
Plane 1	✓	✓	✓	✓
Plane 2	✓	✓	✓	✓

To our surprise, both VGG models were able to identify all of the images we tested from the internet, yet the ResNet models struggled. These results would surely vary based on the images that we selected, but it was encouraging to see the model work with more hands-on application. One of our images was an unusual looking airplane, which we thought may throw off the model—yet the VGG models did an exceptionally good job recognizing our random images. We are not sure why the ResNet models performed so much worse, but this may be due to chance. The ResNet models were also much more likely to label the image as a plane. We would need to run the experiment with a greater sample size to get more meaningful results.

4 Conclusion

In this lab, we used transfer learning to leverage pretrained weights from some of the most popular models in deep learning. By measuring training time, testing time, and classification accuracy, we were able to compare models that used VGG and ResNet architectures for feature learning. Our studies showed that there was little difference in test accuracy, but there were notable costs for the more complex models in terms training and testing times. Overall, this analysis highlights the power of transfer learning as we were able to achieve 81% accuracy after just five epochs of training. These findings underscore the adaptability and efficiency of deep learning techniques which can be used to address a diverse array of real-world image classification challenges.

5 Code Appendix

```
# -*- coding: utf-8 -*-
"""lab4.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/18z8ORDoq99bkoR2cVn7tW5drBFBp8aG-
"""

import torch
import torch.nn as nn
from torch.optim import Adam
from torchsummary import summary
import numpy as np
import matplotlib.pyplot as plt
import time
import progressbar

# Get our VGG and Resnet models
from torchvision.models import vgg16, vgg19, VGG16_Weights, VGG19_Weights
from torchvision.models import resnet50, resnet152, ResNet50_Weights, ResNet152_Weights

# Imports for Data
from torchvision.datasets import CIFAR10
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import transforms, Resize
from torchvision.transforms.functional import to_pil_image
from torchvision.io import read_image

DIMENSION = 224
BATCH_SIZE = 64
DATA_SIZE = 24000
device = 'cuda' if torch.cuda.is_available() else 'cpu'
widgets = [
    ' [',
    progressbar.Timer(format= 'elapsed time: %(elapsed)s'),
    '] ',
    progressbar.Bar('*'),' (',
    progressbar.ETA(), ') ',
]
print(device)

# Set Up Data
preprocess = transforms.Compose([
        transforms.Resize((DIMENSION, DIMENSION)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),])
```

```
48
49  # Load data from CIFAR and trim it down
50  train = CIFAR10('~/data/CIFAR', download=True, train=True, transform=preprocess)
51  test = CIFAR10('~/data/CIFAR', download=True, train=False, transform=preprocess)
52  train = torch.utils.data.Subset(train, list(range(0, DATA_SIZE)))
53
54  # Create data loaders objects
55  train_dl = DataLoader(train, batch_size=BATCH_SIZE, shuffle=True)
56  test_dl = DataLoader(test, batch_size=BATCH_SIZE, shuffle=True)
57
58  # Take a peak at the data
59  print(type(train[0][0]))
60  plt.imshow(train[1][0].cpu().permute(1, 2, 0))
61  print(f"Train length x: {len(train)}, y: {len(test)}")
62  print(f"Current Device: {device}")
63
64  # Training code
65  def train_batch(model, X_train, Y_train, opt, loss_func):
66
67    opt.zero_grad()                       # Flush memory
68    pred = model(X_train)
69    batch_loss = loss_func(pred, Y_train) # Compute loss
70    batch_loss.backward()                 # Compute gradients
71    opt.step()                            # Make a GD step
72
73    return batch_loss.detach().cpu().numpy()
74
75
76  def accuracy(model, test_loader):
77    correct = 0
78    total = 0
79
80    # Set the model to evaluation mode
81    model.eval()
82
83    # Disable gradient calculation
84    print("Computing accuracy...")
85    start = time.time()
86    bar = progressbar.ProgressBar(max_value=len(test_loader), widgets=widgets).start()
87    with torch.no_grad():
88        for i, batch in enumerate(test_loader):
89            inputs, labels = batch
90            inputs = inputs.to(device)  # Send inputs to the device (CPU or GPU)
91            labels = labels.to(device)  # Send labels to the device
92
93            # Forward pass
94            outputs = model(inputs)
95
96            # Get predicted class labels
97            _, predicted = torch.max(outputs, 1)
```

```
98
99              # Update counts
100             total += labels.size(0)
101             correct += (predicted == labels).sum().item()
102             bar.update(i)
103
104     # Calculate accuracy
105     accuracy = (correct / total) * 100.0
106     testing_time = time.time() - start
107     return accuracy, testing_time
108
109
110 def train_model(model, epochs, optimizer, loss):
111
112     # Put model in training mode
113     model.train()
114     losses, is_accuracies, os_accuracies, n_epochs = [], [], [], epochs
115     start = time.time()
116     for epoch in range(n_epochs):
117         print(f"Running epoch {epoch + 1} of {n_epochs}")
118         bar = progressbar.ProgressBar(max_value=len(train_dl), widgets=widgets).start()
119         epoch_losses = []
120
121         # Loop to train each batch and measure its loss value
122         for i, batch in enumerate(train_dl):
123             x, y = batch
124             x, y = x.to(device), y.to(device)
125             batch_loss = train_batch(model, x, y, optimizer, loss)
126             epoch_losses.append(batch_loss)
127             bar.update(i)
128
129         # Log and track epoch progress
130         epoch_loss = np.mean(epoch_losses)
131         losses.append(epoch_loss)
132         print()
133         print(f"Epoch {epoch + 1} had loss of {epoch_loss}")
134
135     training_time = time.time() - start
136     return training_time
137
138 # Code we want to run to test each model
139 def test_model(model):
140     test_accuracy, inference_time = accuracy(model, test_dl)
141     print()
142     print(f"Test Accuracy: {test_accuracy}")
143     print(f"Inference time: {inference_time}")
144
145 # Define and train the vgg models
146 vgg_models = [
147     vgg16(weights=VGG16_Weights.IMAGENET1K_V1).to(device),
```

```
    vgg19(weights=VGG19_Weights.IMAGENET1K_V1).to(device),
]

for model in vgg_models:

  # Disable training for VGG's feature layers
  for param in model.features.parameters():
    param.requires_grad = False

  # Set up the model with our classifier
  model.avgpool = nn.AdaptiveAvgPool2d(output_size=(1,1)).to(device)
  model.classifier = nn.Sequential(
                                  nn.Linear(512,128),
                                  nn.ReLU(),
                                  nn.Dropout(0.2),
                                  nn.Linear(128,10),
                                  nn.Sigmoid()).to(device)
  summary(model, (3, DIMENSION, DIMENSION))

  # Train the model
  loss_func = nn.CrossEntropyLoss().to(device)
  opt = Adam(model.parameters(), lr=1e-3)
  training_time = train_model(model, 5, opt, loss_func)

  # Print some training stats
  print(f"Total training time: {training_time}")

  # Test the model
  test_model(model)

# Define and train the resnet models
res_models = [
    resnet50(weights=ResNet50_Weights.IMAGENET1K_V1).to(device),
    resnet152(weights=ResNet152_Weights.IMAGENET1K_V1).to(device)
]

for model in res_models:

  # Disable training for ResNet's feature layers
  for param in model.parameters():
    param.requires_grad = False

  # Set up the model with our classifier
  model.fc = nn.Sequential(nn.Flatten(),
                           nn.Linear(2048,128),
                           nn.ReLU(),
                           nn.Dropout(0.2),
                           nn.Linear(128,10),
                           nn.Sigmoid()).to(device)
  summary(model, (3, DIMENSION, DIMENSION))
```

```
198
199     # Train the model
200     loss_func = nn.CrossEntropyLoss().to(device)
201     opt = Adam(model.parameters(), lr=1e-3)
202     training_time = train_model(model, 5, opt, loss_func)
203
204     # Print some training stats
205     print(f"Total training time: {training_time}")
206
207     # Test the model
208     test_model(model)
209
210   # Test Some Images from Google
211   # 2 frogs, 2 ships, 2 airplanes
212   classes = {
213       0: "airplane",
214       1: "automobile",
215       2: "bird",
216       3: "cat",
217       4: "deer",
218       5: "dog",
219       6: "frog",
220       7: "horse",
221       8: "ship",
222       9: "truck"
223   }
224   paths = ["frog1.png", "frog2.png", "ship1.png", "ship2.png", "plane1.png", "plane2.png"]
225   for path in paths:
226
227     # Get tensor from file
228     img = read_image(path)
229     img = img[:3, :, :]
230     img = to_pil_image(img)
231
232     # Preprocess the image
233     img = preprocess(img)
234
235     # Plot image
236     plt.imshow(img.permute(1, 2, 0))
237
238     # Check the predicted class
239     models = vgg_models + res_models
240     for model in models:
241       model.eval()
242       prediction = model(img[None, :, :, :].to(device))
243       prediction = classes[torch.argmax(prediction, 1).item()]
244       print(f"Prediction for {path}: {prediction}")
```